CSE 120: Principles of Operating Systems

Lecture 8

Segmentation and Paging

February 6, 2006

Prof. Joe Pasquale

Department of Computer Science and Engineering

University of California, San Diego

© 2006 by Joseph Pasquale
Before We Begin ...

Read Chapter 8 (Main Memory)

Homework Assignment #3
• Due Tuesday, February 7, midnight

Programming Assignment #3
• Due Saturday, February 11, midnight

Midterm Exam
• February 13
• Will cover all material up to that point
Recall: Process Address Space

**Text**: program instructions
- Execute-only, fixed size

**Data**: variables (static, heap)
- Read/write, variable size
- Dynamic allocation by request

**Stack**: activation records (local/auto)
- Read/write, variable size
- Automatic growth/shrinkage
Fitting Process Into Physical Memory

Need to find a large enough hole

May not succeed (even if enough fragment space)

Even if successful, still inefficient
  • Space must be allocated for potential growth areas

Solution: break process into pieces
  • Distribute into available holes

Process Memory

Physical Memory

Text

Data

Stack
Fitting Process Into Physical Memory

Need to find a large enough hole

May not succeed (even if enough fragment space)

Even if successful, still inefficient
  • Space must be allocated for potential growth areas

Solution: break process into pieces
  • Distribute into available holes
Two Approaches

Segmentation
- Break process up into logical segments
- Segments are generally of different sizes

Paging
- Break process up into physical pages
- Pages are the same size
Segmented Address Space

Address space is a set of segments

Segment: a linear address memory
- Typically contains logically-related information
- Examples: program code, data, stack

Each segment has an identifier $s$, and a size $N_s$
- $s$ between 0 and $S-1$, $S = \text{max number of segments}$

Logical addresses are of the form $(s, i)$
- Offset $i$ within segment $s$, $i$ must be less than $N_s$
Example: Segmented Address Space

Segment 0
Text
(--x)
0, 0
0, N₀-1

Segment 1
Data
(rw-)
1, 0
1, N₁-1
2, M₁-N₂

Segment 2
Stack
(rw-)
2, M-1

Segment 3
Shared Data
(r--)
3, 0
3, N₃-1

Nₙ = size of segment s
M = max segment size
Segment-based Address Translation

Problem: how to translate

- a segment-based logical address \((s, i)\)
- into physical address \(x\)?

Solution: use a segment (translation) table

- Translate \((s, i)\) into physical address \(x = t(s, i)\)
Segment Table

One per process (typically)

Table entry elements

- V: valid bit
- Base: segment location
- Bound: segment size
- Perm: permissions

Located in memory

- Segment table base register
- Segment table size register
Address Translation

Physical address = segment base of s + offset i
BUT, first check validity
Check if Segment Number $s$ is Valid

Logical Address

Segment $s$  Offset $i$

STBR  STSR

$s < STSR$

V Base Bound Perm ...

Physical Address
Check if Segment Entry $s$ is Valid

Logical Address

Segment $s$  Offset $i$

STBR  STSR

$V == 1$

$V$ Base Bound Perm ...

Physical Address
Check if Offset $i$ is Within Bounds

Logical Address

Segment $s$  Offset $i$

STBR  STSR

$i < \text{Bound}$

V Base Bound Perm ...

Physical Address
Check if Operation is Permitted

Logical Address

Segment $s$    Offset $i$

STBR
STSR

Perm (op)

V Base Bound Perm ...

Physical Address
Translate Address

Logical Address

Segment $s$  Offset $i$

STBR  STSR

V  Base  Bound  Perm ...

Physical Address
### Sizing the Segment Table

#### Logical Address

<table>
<thead>
<tr>
<th>Segment s</th>
<th>Offset i</th>
</tr>
</thead>
</table>

- **Number of bits** $n$ specifies max size of table, where number of entries = $2^n$
- **V**
- **Base**
- **Bound**
- **Perm**
- **...**

- **Number of bits** needed to address physical memory
- **Number of bits** needed to specify max segment size

- **Number of bits** $n$ specifies max size of segment
Example of Sizing the Segment Table

Given 32 bit logical address, 1 GB physical memory (max)

- 5 bit segment number, 27 bit offset

Segment s: 5 bits
Offset i: 27 bits

5 bits to address
$2^5 = 32$ entries

30 bits needed to address 1 GB

8 bytes needed to contain 61 (1+30+27+3+...) bits

Table size = $32 \times 8 = 256$ bytes

27 bits needed to size up to 128 MB

Table size: 32 x 8 = 256 bytes
Pros and Cons of Segmentation

**Pro: Each segment can be**
- Located independently
- Separately protected
- Grown/shrunk independently

**Pro: Segments can be shared between processes**

**Con: Variable-size allocation**
- Difficult to find holes in physical memory
- External fragmentation
Paged Address Space

Logical (process) memory
- Linear sequence of pages

Physical memory
- Linear sequence of frames

Pages and frames
- Frame: a physical unit of information
- A page fits exactly into a frame
- Fixed size, all pages/frames same size
Page-based Logical Addressing

Logical addresses are of form \((p, i)\)

- \(p\) is page number, 0 to \(N_L-1\)
- \(i\) is offset within page
- Note: \(i\) is less than page size
  - no need to check

Size of logical address space

- \(N_L\) = max number of pages
- \(N_L \times\) page size = size of logical address space
Frame-based Physical Addressing

Physical addresses are of form \((f, i)\)

- \(f\) is frame number, 0 to \(N_p-1\)
- \(i\) is offset within frame
- Note: \(i\) is less than frame size
  - since page size = frame size

Size of physical address space

- \(N_p = \text{max number of frames}\)
- \(N_p \times \text{frame size} = \text{size of physical address space}\)
Page-based Address Translation

Problem: how to translate
  • a page-based logical address \((p, i)\)
  • into physical address \((f, i)\)?

Solution: use a page (translation) table
  • Translate page \(p\) into frame \(f = t(p)\)
Logical Pages to Physical Frames

Each page of logical memory corresponds to entry in page table.

Page table “maps” logical page into frame of physical memory.
Page Table

One per process (typically)

Table entry elements
- V: valid bit
- R: reference bit
- M: modified bit
- Frame: page location

Located in memory
- Page table base register
- Page table size register
Address Translation

Physical address = frame of p + offset i

BUT, first check validity

Logical Address

Page p Offset i

Physical address = frame of p + offset i

BUT, first check validity

Physical Address
Check if Page Number $p$ is Valid

Logical Address

Page $p$  Offset $i$

PTBR  PTSR

$p < PTSR$

V | R | M | Frame | Perm | ...

Physical Address

27
Check if Page Table Entry p is Valid

Logical Address

Page p  Offset i

PTBR  PTSR

V == 1

V | R | M | Frame | Perm | ...
Check if Operation is Permitted
Translate Address

Logical Address

Page $p$  
Offset $i$

PTBR
PTSR

V | R | M | Frame | Perm |
...

concat

Physical Address
Physical Address by Concatenation

Logical Address

Page p  Offset i

PTBR  PTSR

VRM Frame Perm ...

Physical Address

Frame f  Offset i
Sizing the Page Table

Logical Address

Page p | Offset i

Number of bits $n$ specifies max size of table, where number of entries = $2^n$

Number of bits $n$ specifies page size

Number of bits needed to address physical memory *in units of frames*
Example of Sizing the Page Table

Given 32 bit logical address, 1 GB physical memory (max)

- 20 bit page number, 12 bit offset

Page p: 20 bits

Offset i: 12 bits

20 bits to address
$2^{20} = 1 \text{ M entries}$

18 bits to address
$2^{30}/2^{12} \text{ frames}$

4 bytes needed to contain
$24 (1+1+18+3+...) \text{ bits}$

Page size =
frame size =
$2^{12} = 4096 \text{ bytes}$

Table size =
$1 \text{ M x 4} = 4 \text{ MB}$
Multi-Level Page Tables

Logical Address

1st-level Page d | 2nd-level Page p | Offset i

Points to (base) frame containing 2nd-level page table

Physical Address
Segmentation vs. Paging

Segment is good “logical” unit of information
  • Can be sized to fit any contents
  • Makes sense to share (e.g., code, data)
  • Can be protected according to contents

Page is good “physical” unit of information
  • Simple memory management

Why not have best of both
  • Segmentation on top of paging
Combining Segmentation and Paging

Logical memory composed of segments

Each segment composed of a set of pages

Segment table: maps each segment $s$ to page table

Page tables (like before)
Address Translation

Logical address: [segment s, page p, offset i]

Do various checks

- $s < \text{STSR}$, $V == 1$, $p < \text{bound}$, perm (op)
- May get a segmentation violation

Use segment s to index segment table to get page table

Use page p to index page table to get frame f

Concatenate frame and offset to get physical address
Segment/Page Address Translation

Logical Address

Segment $s$  Page $p$  Offset $i$

V Base Bound ...

$p < \text{bound}$

Page table of segment

V R M Frame ...

concat

Physical Address
Cost of Translation

Each page table lookup costs a memory reference
  • For each reference, additional references required
  • Slows machine down by factor of 2 or more

Take advantage of locality of reference
  • Most references are to a small number of pages
  • Keep translations of these in high-speed memory

Problem: we don’t know which pages until referenced
Translation Lookaside Buffer (TLB)

Fast associative memory keeps most recent translations
If key matches, get frame number for physical address, else wait for normal memory translation (in parallel)
Translation Cost with TLB

Cost is determined by

- Speed of memory: ~ 50 nsec
- Speed of TLB: ~ 10 nsec
- Hit ratio: fraction of refs satisfied by TLB, ~95%

Speed with no address translation: 50 nsec

Speed with address translation

- TLB miss: 100 nsec (100% slowdown)
- TLB hit: 60 nsec (20% slowdown)
- Average: \(60 \times .95 + 100 \times .05 = 62\) nsec
TLB Design Issues

The larger the TLB
  • the higher the hit rate
  • the slower the response
  • the greater the expense

TLB has a major effect on performance!
  • Must be flushed on context switches
  • Alternative: tagging entries with PIDs

MIPS: has only a TLB, no page tables!
  • Devote more chip space to TLB